Le modèle VLISP : description, implémentation et évaluation

Jérôme CHAILLOUX

Octobre 1980

Notre étude présente la réalisation de *trois systèmes* VLISP (dialecte du langage LISP) développés à l'université de Paris 8 - Vincennes. Ces systèmes ont été réalisés :
- sur micro-processeur à mots de 8 bits (Intel8080/Zilog80)
- sur mini-ordinateur à mots de 16 bits (PDP11)
- sur gros ordinateur à mots de 36 bits (PDP10)

De ces réalisations est extrait un ***modèle d'implémentation.***

Cette étude propose des solutions aux problèmes de construction et d'évaluation de tels systèmes. Ces problèmes sont :

1) La description exhaustive des implémentations.
   Nous proposons une description fondée sur la *machine référentielle* VCMC2.

2) La représentation adéquate des objets et des fonctions VLISP.
   Nous avons associé des ***propriétés naturelles*** aux objets dès leur création et nous avons établi une ***typologie fonctionnelle*** de ces objets.

3) L'efficacité de l'interprète (en place, en temps d'exécution et en compréhension). Notre interprète effectue, pour ses besoins propres, une allocation optimale de la mémoire (allocation mesurée en terme d'appels du module CONS). L'accès direct (ne nécessitant qu'un accès mémoire) aux valeurs des objets de type variable et fonction, et la classification des fonctions par types permettent un ***lancement immédiat*** de toutes les fonctions. La transparence de nos méthodes de description est une conséquence naturelle des deux choix précédents.

4) Le ***pouvoir*** des structures de contrôle.
   Notre modèle d'implémentation généralise les structures de contrôle de VLISP **SELF** et **ESCAPE** en intégrant les nouvelles constructions **EXIT**, **WHERE** **et** **LETF** et en unifiant totalement leur description et leur implémentation.

5) La souplesse d'utilisation du système.
   Nous introduisons le nouveau concept de ***CHRONOLOGIE*** qui permet de créer dynamiquement de nouveaux évaluateurs, permettant les traces méta-circulaires.

Une incarnation de notre modèle est donnée dans [CHAILLOUX 80], sous la forme de la réalisation d'un système VLISP pour la machine référentielle VCMC2.

1.0 HISTORIQUE CRITIQUE DE L'IMPLEMENTATION DE LISP.

Le langage LISP [BERKELEY 74, WEISSMAN 67, ALLEN 78] est aujourd'hui le langage le plus utilisé dans les domaines de l'Intelligence Artificielle [WINSTON 77] et de la théorie de la programmation [NIVAT 79]. La particularité majeure de LISP est d'avoir pu évoluer de façon naturelle depuis sa naissance en 1960 [McCARTHY 60a, 60b], à mesure qu'apparaissaient de nouveaux besoins. L'échec patent des rares tentatives de standardisations *{Note 1}* a d'une part permis sa survie et d'autre part conduit à une amélioration constante du langage tant en efficacité à l'interprétation qu'en pouvoir de ses concepts.

Il en résulte aujourd'hui une floraison d'implémentations, bien loin du premier LISP 1 [McCARTHY 60a], ayant chacune des spécifications particulières en fonction de leur utilisation et du moment de leur création.

On distingue ainsi 4 grandes familles d'implémentations :

1) La lignée LISP 1.5 [McCARTHY 62]. Descendante directe de LISP 1 qui était un système interactif (avec le "FLEXOWRITER system"), elle a donné le LISP 1.6 [QUAM 72] très proche de LISP 1.5, mais ayant abandonné la liaison par A-liste, décision dont les conséquences ont été monumentales. Par la suite, l'importance capitale de l'intégration des instruments de mise au point devint de plus en plus évidente, et LISP 1.6 fut augmenté d'un éditeur et d'aides à la mise au point pour donner le U.C.I. LISP [BOBROW 73a].

2) Les systèmes INTERLISP (anciennement BBN LISP) [TEITELMAN 75] orientés sur une utilisation exclusivement interactive, et qui ont donnés naissance à des sous-systèmes tels que CLISP [TEITELMAN 73] ou DLISP [TEITELMAN 77] qui font une utilisation intensive de périphériques interactifs tels les écrans à haute résolution.

3) MACLISP [MOON 74, LAUBSCH 76] au M.I.T. dont le développement s'est réalisé en symbiose avec celui du système MACSYMA [MACSYMA 75] et qui a inspiré la conception de la machine LISP du M.I.T. [WEINREB 79].

4) Enfin les systèmes VLISP [GREUSSAY 76a, CHAILLOUX 78a, CHAILLOUX 80] développés en France à l'Université de Paris 8 - Vincennes, implantés sur les 3 catégories de systèmes matériels (gros, mini et micro) et qui sont dotés des structures de contrôle les plus puissantes.

Le point commun de ces familles d'implémentations est d'avoir toutes une réalisation sur l'ordinateur de référence PDP-10 [DEC 78a], qui est la machine la plus utilisée dans le cadre des recherches en Intelligence Artificielle. Pour cette raison les performances de ces différentes réalisations sont aisément mesurables.

En plus du PDP-10, ces systèmes LISP n'ont été implantés que sur un nombre restreint d'autres matériels en général de taille très importante tels l'IBM série 360/370 [BOLCE 68, HAFNER 74], l'UNIVAC 1100 ou l'IRIS-80 pour lequel on dénombre 3 réalisations en France, le TLISP à l'Université Paul Sabatier de TOULOUSE [DURIEUX 78], le RLISP au laboratoire IMAG de GRENOBLE [LUX 78]

{Note 1} ces standards sont aussi prématurés qu'arbitraires. Prématurés car ils tombent en désuétude aussitôt créés (voir le standard de [HEARN 69] puis [MARTI 79]) et ne tiennent pas compte de l'évolution du langage, arbitraires car ils sont essentiellement définis pour satisfaire un programme particulier (REDUCE 2 dans le cas de HEARN [HEARN 73, 74]).

et SIRLISP à l'E.N.S.T de Paris [COILLAND 79].

LISP 1.5 et LISP 1.6, hormis des réalisations-jouets à fins pédagogiques, sont à présent tombés en désuétude.

MACLISP et INTERLISP, trop dépendant de la machine PDP10 (et même du système d'exploitation utilisé respectivement ITS et TENEX) n'ont pu être transportés dans leur intégralité sur d'autres machines *{Note 1}*.

Seul, VLISP, (du fait probablement de sa conception en Europe et de la diversité des ordinateurs qui y sont disponibles) a pu (ou a dû) être réalisé sur un grand nombre de machines très variées telles que CAE510 [GREUSSAY 72], CAB500 [WERTZ 74], T1600 [GREUSSAY 75], PDP10 [CHAILLOUX 78c], SOLAR16 [GREUSSAY 78b], 8080 et Z80 [CHAILLOUX 79a] et PDP11 [GREUSSAY 79b].

La diversité de toutes ces réalisations a entrainé la naissance d'histoires (et d'historiens) de LISP, apportant de précieuses informations sur <u>l'évolution naturelle</u> des langages de programmation *{Note 2}* [McCARTHY 78], [WHITE 78], [STOYAN 78a,78b].

Les recherches actuelles de construction des systèmes LISP s'élaborent pour l'essentiel selon les 4 axes suivants :

- implémentation sur des ordinateurs universels classiques tel le VAX-11 [DEC 78e] ou le S-1 [HAILPERN 79].
- implémentation sur des ordinateurs spécialisés, voire des micro-ordinateurs, à ressources limitées [CHAILLOUX 78a, TAFT 79].
- réalisation de machines spécialisées [GREENBLATT 74, KNIGHT 74, SHIMADA 76, LISPMACHINE 77, LECOUFFE 77, TAKI 79, WEINREB 79].
- apparition d'unités centrales, en technologie V.L.S.I. [MEAD 80], capables d'interpréter directement LISP à une translation près du langage source en sa représentation arborescente, dont le répertoire des sommets constitue un jeu d'instructions d'interprétation directe de LISP (les CHIP-LISP) [STEELE 79, HOLLOWAY 80].

{Note 1} En date de Septembre 1978, MACLISP ne tournait encore qu'avec beaucoup de difficultés, en ce qui concernait ses aspects-système évolués, au laboratoire d'Intelligence Artificielle de Stanford, sous système WAITS [GREUSSAY 78a].

{Note 2} Par opposition aux cas de langages à génération spontanée ou à comités [HORNING 79], à définition discrétionnaire.

2.0 COMMENT DECRIRE LES IMPLEMENTATIONS : la machine VCMC2.

Dès son apparition LISP a été exposé méta-circulairement *{Note 1}* afin de décrire ses propres implémentations.
Cette utilisation de LISP, très commode, ne satisfaisait pas les implémenteurs experts, qui n'avaient à leur disposition que ces descriptions méta-circulaires accompagnées, dans le meilleur des cas, du texte d'une implémentation particulière. Tous les problèmes réels d'implémentation en machine étaient soit totalement ignorés (dans le cas où LISP était utilisé à sa propre description) soit entièrement orientés vers une machine particulière.

Pour illustrer cette situation, nous utiliserons dans cette section, une succession progressive de descriptions du module classique EVLIS : nous partirons de sa description méta-circulaire en LISP, nous la qualifierons sur des machines particulières, enfin nous en donnerons la description dans notre modèle de référence.

Voici la description de la fonction EVLIS, en LISP classique, telle quelle est donnée dans les manuels de référence :

LISP classique

```
(DE EVLIS (L)
   (COND
     ((NULL L) NIL)
     (T (CONS (EVAL (CAR L)) (EVLIS (CDR L))))))
```

Une telle description purement méta-circulaire était déjà remise en question par John McCARTHY, le créateur du langage : il introduisit deux langages LISP, un langage algorithmique (le Méta-langage) qui utilisait des M-expressions et un langage de programmation (le LISP tel qu'on le parle encore) qui mettait en jeu les S-expressions [McCARTHY 62]. Ce double langage permettait de distinguer les données décrites en S-expression des programmes décrits sous forme de M-expressions [PERROT 79].

{Note 1} LISP est spécifiable par un interprète dit méta-circulaire : un tel interprète, écrit en VLISP, ramasse en une seule description la spécification du langage et celle de l'interprète lui-même (voir la description de l'évaluateur VLISP-10 en VLISP-10 donné dans [CHAILLOUX 78c] page 20).

La traduction du langage algorithmique vers le langage de programmation LISP devait s'effectuer automatiquement *{Note 1}*. on sait aujourd'hui que seul le langage de programmation a survécu.

Voici donc la description de la fonction EVLIS en utilisant la notation sous forme de M-expressions :

M-expressions

```
evlis[l]=[null[l]→NIL;T→cons[eval[car[l]];evlis[cdr[l]]]]
```

Si on considère qu'une implémentation particulière constitue une description de référence on se heurte aux deux difficultés que sont :

1) l'opacité inhérente à la méconnaissance préliminaire de l'ordinateur source

2) la non-généralité chronique de ce type de description.

Voici donc cette même fonction telle qu'elle est codée dans le langage machine du PDP10 [CHAILLOUX 78c] :

Langage machine PDP-10

```
EVLIS:
     JUMPE   A1,VPOPJ        ; pas d'argument
     HRRZ    A2,MEM(A1)      ; A2 ← (CDR A1)
     PUSH    P,A2            ; qui est sauvé
     PUSHJ   P,EVALCA        ; évalue la 1ère val.
     HLRZ    A1,MEM(A1)      ; A1 ← (CONS A1)
     EXCH    A1,MEM(FREE)
     EXCH    FREE,A1
     POP     P,A2            ; récupère le reste
     CAMGE   A2,BLIST        ; au moins 2 ?
     JRST    VPOPJ           ; non : fini
     PUSH    P,A1            ; sauve le 1er doublet
     PUSH    P,A1            ; sauve le dernier
     PUSH    P,A2            ; sauve le reste
     MOVEI   A1,(A2)         ; A1 ← les arguments
EVLIS1:
     HRRZ    A2,MEM(A1)      ; A2 ← (CDR A1)
     MOVEM   A2,(P)          ; dans le sommet de la pile
     PUSHJ   P,EVALCA        ; évalue l'argument suivant
     HLRZ    A1,MEM(A1)      ; A1 ← (CONS A1)
```

{Note 1} c'est l'absence des caractères spéciaux [,], et → sur la perforatrice de cartes IBM026 qui a empéché l'utilisation systématique des M-expressions.

```
21        EXCH    A1,MEM(FREE)
22        EXCH    FREE,A1
23        MOVE    A2,-1(P)          ; A2 ← le dernier doublet
24        HRRM    A1,MEM(A2)        ; (RPLACD A2 A1)
25        MOVEM   A1,-1(P)          ; sauve le dernier doublet
26        MOVE    A1,(P)            ; A1 ← le reste
27        CAML    A1,BLIST          ; il reste des éléments ?
28        JRST    EVLIS1            ; oui.
29        SUB     P,[3,,3]          ; non : nettoie la pile .
30        MOVE    A1,1(P)           ; A1 ← 1er doublet
31    VPOPJ:
32        POPJ    P,                ; et voilà.
```

Le point 1) est illustré ici par l'instruction MOVEI, utilisée à la ligne 15, qui emploie classiquement la propriété d'adressage suivante :

immédiat + indexé = direct

et permet un gain substanciel de temps :

Temps d'exécution des instructions		
MOVE	A1,A2	1.14 micro-sec
MOVEI	A1,(A2)	0.62 micro-sec

sur l'unité centrale PDP 10 KI [DEC 78a].

On notera également la confusion conceptuelle introduite par l'utilisation du pointeur de pile P comme registre d'index, pointeur qui ira même (voir la ligne 30) jusqu'à être employé à l'extérieur de la zone pile.

La non-généralisation de cette description est malheureusement une conséquence de l'utilisation raisonnablement experte d'une machine particulière.

Nous devons aller au delà de ces 3 représentations pour aborder, décrire correctement, résoudre enfin les véritables problèmes d'implémentation en machine :

- la représentation en LISP classique montre assez bien l'intention de la fonction mais ignore complétement les problèmes d'allocation des ressources, souvent limitées, ainsi que la gestion de la récursion.

- la représentation en M-expressions tout en évoquant approximativement une notation algébrique présente les mêmes inconvénients que la représentation en LISP et introduit un niveau supplémentaire de traduction.

- la représentation en langage machine PDP10 fait surgir de nouveaux problèmes locaux tels que l'accès aux demi-mots de la machine voire l'utilisation du pointeur de pile comme registre d'index. La description en langage machine est alors supplantée par l'expérience nécessaire à la mise en jeu correcte d'adressages et de jeux d'instructions particuliers.

Nous proposons donc de représenter l'implémentation au moyen de programmes d'une machine de référence, la machine VCMC2, descendante directe de la machine VCMC1 [CHAILLOUX 78b], et indépendante des représentations internes des objets eux-mêmes. C'est de cette description que naissent naturellement les incarnations opérationnelles particulières de VLISP sur les ordinateurs les plus diversifiés. Cette machine, décrite dans [CHAILLOUX 80], est simulée en VLISP-10 sur ordinateur PDP-10.

La description de VLISP très concise, qui en découle dégage les caractéristiques fondamentales de l'implémentation de modules tels que EVLIS en machine, i.e. :

1) les actions de base (test par rapport à NIL, CDR, CONS ...)

2) la gestion de la récursion par utilisation d'une pile et d'un accumulateur.

3) le mécanisme du sauvetage et de la restauration du reste de la liste et du résultat intermédiaire au travers de la pile.

4) l'internalisation totale des fonctions interfaces : entrées/sorties et traces (ce que ne pouvait pas faire la notation FILTRE de [GREUSSAY 76b]).

Voici donc la description, en VCMC2, de l'implémentation en machine de la fonction EVLIS :

La fonction EVLIS décrite en VCMC2

```
EVLIS:  TNIL     A1,,[RETURN]
        CDR      A1,TST,[CALL (EVCAR)]
        XTOPST   A1,,[CALL (EVLIS)]
        CONS     TST,A1,[RETURN]
```

Cette représentation générale atteint le degré de précision suffisant pour :

1) appréhender la véritable structure en actions de base de l'implémentation (ce que ne pouvait pas faire la description d'INTERLISP *{Note 1}*

{Note 1} la notation utilisée dans la description de [MOORE 76] ressemble à la notation sous forme de M-expressions et ne rend pas compte de la gestion de la récursion et des résultats intermédiaires. Sachant que EVLIS est la version SUBR du module LIST (de type FSUBR) [MOORE 76 pp. 11] ne peut décrire cette dernière qu'elliptiquement par le dispositif typographique des points de suspension :

LIST[x1;x2;...xk]
Return CONS[x1;CONS[x2;...CONS[xk;NIL]...]]

2) expliciter totalement la gestion de la pile de récursion en unifiant dans l'adressage la mise en jeu des accumulateurs et du cache de pile.

3) indiquer la circulation dans la pile des états d'évaluation intermédiaires du contenu initial du registre A1

4) préciser enfin la structure de contrôle par attachement à chaque instruction d'une continuation complexe.

3.0 EFFICACITE ET PUISSANCE DE L'INTERPRETATION.

Du fait de sa représentation des objets et des fonctions, notre modèle permet de réduire considérablement le temps d'interprétation et de disposer d'un grand nombre de structures nouvelles, qui facilitent l'écriture des programmes, en améliorent la lisibilité et en diminuent sensiblement la taille.

Voici la définition de la fonction de FIBONACCI utilisée pour des comparaisons de vitesse sur plusieurs interprètes LISP fonctionnant sur PDP10 :

```
(DE FIB (n)
    (COND
       ((ZEROP n) 1)
       ((EQ n 1) 1)
       (T (PLUS (FIB (SUB1 n)) (FIB (DIFFER n 2))))))

(FIB 20) ☞ 10946
```

Voici les temps relevés sur MACLISP, INTERLISP, VLISP-10 et VLISP-11 *{Note 1}* pour le calcul de l'appel **(FIB 20)** :

[Temps en secondes]			
MACLISP	INTERLISP	VLISP-10	VLISP-11
48.66	105.78	13.34	35.2

La rapidité remarquable de l'interprétation de notre modèle est dûe :

1) à l'utilisation intensive des propriétés naturelles des atomes et principalement du type associé à chaque fonction. Le type d'une fonction LISP est déterminé par le langage dans lequel elle est écrite, le mode de passage de ses arguments (les arguments sont ou ne sont pas évalués) ainsi que le nombre de ses arguments. Cette typologie est abondamment décrite dans [CHAILLOUX 80 chapitre 4]. Ce type est directement accessible, ce qui permet de réaliser en même temps et le test du type et l'appel de la routine associée, au moyen d'un seul branchement indirect indexé. En terme de syntaxe, VLISP utilise donc l'appel par nom des

{Note 1} MACLISP utilise dans le cadre de ce test une unité centrale PDP10 KA-10 et des mémoires à 1.8 micro-secondes [WHITE 76], INTERLISP une unité centrale PDP10 KA-10 et des mémoires à 1.0 micro-secondes, VLISP-10 une unité centrale PDP10 KI-10 et des mémoires à 1.0 micro-secondes. VLISP-11 n'a à sa disposition qu'une micro-unité centrale LSI 11/02.

fonctions.

2) au fait que l'interprète ne réalise plus <u>aucun CONS</u> pour ses besoins propres. L'utilisation exclusive d'une pile commune de données et de contrôle évite de créer des structures intermédiaires (sous forme de cellules de liste) durant le processus d'évaluation. Le nombre de récupérations de la mémoire dynamique se trouve ainsi réduit aux seuls besoins de l'utilisateur *{Note 1}*.

3) à l'interprétation itérative des fonctions récursives de type NPR et CPR [GREUSSAY 76c].

4) à l'utilisation des structures de contrôle de VLISP qui permettent une écriture plus concise et une interprétation plus rapide. En particulier :

   - l'opérateur point-fixe du λ-calcul [ROBINET 78], représenté par la fonction **SELF** de [GREUSSAY 77]

   FIBONACCI se reécrit alors commodément :

```
(DE FIB (n)
     (IF (ZEROP n) 1
         (IF (EQ n 1) 1
             (PLUS (SELF (SUB1 n))
                   (SELF (DIFFER n 2))))))
```

   - les fonctions d'échappement **ESCAPE**. Introduites par [GREUSSAY 76b], elles permettent d'associer dynamiquement des fonctions d'échappement à des atomes, réalisant ainsi des sorties non-locales.

5) enfin à de nouvelles structures de contrôle. Ces nouvelles structures de contrôle vont pouvoir réaliser :

   - la description de <u>fonctions anonymes</u> (i.e. qui ne sont pas associées à des noms) en utilisant la forme **INTERNAL**.

   - les sorties locales à une fonction en utilisant la fonction **EXIT**.

   - la redéfinition dynamique de fonctions de n'importe quel type en utilisant la primitive **WHERE**.

   - la définition d'une nouvelle classe de fonctions, les variables fonctions, apportant une solution nouvelle au problème des variables internes.

{Note 1} La gestion de la mémoire en LISP est automatique et dynamique ce qui assure une utilisation optimum de la place mémoire mais oblige à construire des modules d'accès et d'allocation (le module CONS) spécialisés ainsi qu'un récupérateur de mémoire, dispositif qui ralentit l'interprète.

3.1 Description de fonctions anonymes

Notre modèle permet de décrire des fonctions anonymes (non associées à des symboles atomiques) de n'importe quel type au moyen d'une des deux formes **INTERNAL** suivantes :

```
(INTERNAL type adresse-mémoire)
(INTERNAL type fonction)
```

La première forme permet de décrire des fonctions écrites en langage machine, et la seconde des fonctions écrites en VLISP.

Ces deux nouvelles formes généralisent l'ancienne utilisation de la forme LAMBDA (qui a été gardée par souci de compatibilité et de simplification de l'écriture des EXPR anonymes) pour tous les types de fonctions et permettent ainsi de limiter l'utilisation des noms associés aux fonctions.

Cette limitation des noms réduit d'autant plus l'espace mémoire nécessaire au stockage des symboles atomiques et accroit la lisibilité du programme qui ne contient plus que les noms indispensables.

3.2 Sorties locales

La forme **EXIT** permet de sortir de la dernière fonction invoquée de type EXPR, FEXPR ou MACRO. La valeur retournée est évaluée dans l'environnement précédant exactement le retour de cette fonction donc évaluée en position terminale. Cette propriété est utilisée pour forcer un traitement de récursion terminale *{Note 1}* ce qui n'était pas le cas des fonctions **RETURN** des anciennes formes **PROG** *{Note 2}*.

Voici la méta-description de la fonction **MEMBER** qui utilise la structure de contrôle itérative **WHILE**.

{Note 1} Ces appels récursifs terminaux sont interprétés d'une manière itérative beaucoup plus efficiente.

{Note 2} La forme **PROG** *a été abandonnée dans notre modèle (ainsi que ses fonctions associées* **GO**, **GOTO** *et* **RETURN***) car les nouvelles fonctions de sorties locales et non-locales sont à la fois plus rapides (en effet le balayage préliminaire de tous les corps de* **PROG** *pour construire la table des étiquettes a disparu) et plus générales (avec la possibilité de retourner directement d'un nombre quelconque d'appels).*

```
(DE MEMBER (a l)
   (WHILE L
      (IF (EQUAL (CAR l) a)
         (EXIT l)
         (NEXTL l))))
```

L'appel de la fonction **EXIT** permet une sortie extraordinaire du **WHILE**.

3.3 Définitions dynamiques de fonctions

La forme WHERE permet de définir dynamiquement de nouvelles fonctions et de rendre fluides *{Note 1}* les fonctions elles-même.

Voici la syntaxe de cette nouvelle forme :

```
(WHERE (<nom> <fval>)  <s1> ... <sN>)
```

Le premier argument du **WHERE** est une définition temporaire d'une fonction **<fval>** associée au nom **<nom>**. Le reste des arguments du WHERE i.e. **<s1>** ... **<sN>** est un corps dans lequel la définition précédente est active. Au sortir du **WHERE**, la définition associée au nom **<nom>** disparait et sa définition antérieure (s'il en possédait une) est restaurée.

Voici la fonction de traçage d'une courbe Dragon [GARDNER 67, SCHOETTL 75] en LOGO-LISP [WERTZ 79] dans laquelle la fonction **tourne** est redéfinie dynamiquement.
Les fonctions **<avance>**, **<droite> et <gauche>** sont des primitives LOGO :

```
(DE tourne () (<droite> 90))

(DE dragon (n l)
   (IF (ZEROP n)
      (<avance> l)
      (WHERE (tourne '(() (<gauche> 90)))
         (dragon (SUB1 n) l))
      (tourne)
      (WHERE (tourne '(() (<droite> 90)))
         (dragon (SUB1 n) l))))
```

{Note 1} Une variable fluide est une variable libre liée dynamiquement.

3.4 Les variables fonctions

Notre modèle propose une solution au problème de l'accès aux variables internes *{Note 1}* de l'interprète. En effet, un choix crucial d'implémentation se pose.
Il n'est que trop tentant d'utiliser des variables VLISP pour accèder à ces variables internes. Appelons OBASE la variable interne qui contient à tout moment la base de numération des nombres en sortie. Une affectation incorrecte de cette variable (par exemple avec une valeur négative), détruirait toute possibilité future d'impression de nombres.
Une deuxième stratégie consiste à utiliser une fonction associée à chaque variable interne. Cette fonction va contrôler la validité des valeurs affectées à la variable. C'est ce qu'on entend par variabilisation ou fonctionnalisation de l'accès.

C'est cette dernière stratégie qui a été adoptée dans notre modèle. De plus celui-ci propose une nouvelle classe de fonctions, les variables fonctions, qui permettent dans une stratégie de fonctionnalisation des variables internes de lier dynamiquement les valeurs de ces variables internes, retrouvant ainsi toutes les propriétés des variables.
En lecture ces variables fonctions n'ont pas d'argument et en écriture elles possèdent un argument (évalué) qui est la nouvelle valeur de la variable fonction.

```
(variable-fonction)           :  lecture
(variable-fonction  valeur)   :  écriture
```

La nouvelle structure **LETF** réalise la liaison dynamique des variables fonctions. Elle possède une syntaxe identique à la macro **LET** :

```
(LETF (variable-fonction  valeur) corps-du-LETF)
```

{Note 1} rappelons que ces variables internes sont les mots mémoires de travail de l'interprète et qu'il est souhaitable que l'utilisateur ait accès à certaines de ces variables pour faciliter l'utilisation du système et en augmenter la puissance.

Cette forme est exécutée en 4 temps :

1) appel de la variable fonction en lecture (sans argument). La valeur retournée est conservée.

2) appel en écriture de la variable fonction avec l'argument transmis au **LETF**.

3) exécution en séquence du corps du **LETF** qui calcule la valeur retournée par la forme **LETF**.

4) re-appel en écriture de la variable fonction avec en argument son ancienne valeur conservée en 1).

Ce type de liaison (de même que celle des variables) est réalisée automatiquement et en particulier le point 4) est effectuée en cas de sortie extraordinaire par un **EXIT** ou un **ESCAPE** enveloppant.

L'utilisation de ces variables fonctions permet donc d'associer à des variables des fonctions qui réalisent des contrôles d'accès dynamiques tant en lecture qu'en écriture.

Voici un exemple d'utilisation de la variable fonction **OBASE** qui contrôle la base de conversion des nombres en sortie. La fonction **PRINT-base-x** imprime son 2ème argument **n** dans la base de numération **x** fournie en 1er argument :

```
(OBASE)     ☞   10

(DE PRINT-base-x (x n)
    (LETF (OBASE x)
        (PRINT n)))

(PRINT-base-x 16 1000)   ☞   3E8

(OBASE)     ☞   10
```

4.0 LA SOUPLESSE D'UTILISATION.

Un certain type de programmation-système consiste à implanter, valider (ou invalider), mesurer, modéliser de nouveaux types d'architectures logicielles ou matérielles (par simulateur), à la limite de l'élaboration de modèles en Intelligence Artificielle.
Cette programmation se fera tout naturellement dans les conditions les plus interactives et les plus expérimentales possibles, et nécessitera des outils permettant la réalisation la plus rapide du modèle à valider. Cette activité demande donc un langage puissant, très facilement et très rapidement mis en oeuvre, car les temps de vie des différents modèles sont extrêmement brefs et soumis à des modifications incessantes.

Ainsi notre système ne mésestime pas l'expertise de ses utilisateurs. Il s'agit d'un système utilisable et utilisé, illustrant la nécessité, parfois sous-estimée, de ne pas entraver l'expertise raisonnable de ses utilisateurs (par exemple en ne l'encombrant pas de contrôles statiques contraignant à une programmation médiocre) mais, tout au contraire, de leur fournir le maximum de pouvoir et de souplesse expressive.

L'utilisation réelle d'une tel système nécessite la faculté de contrôler par niveau le fonctionnement de l'interprète, ce qui amène tout naturellement à l'idée d'évaluateurs multiples selon les circonstances du calcul. Les outils de contrôle de chronologie, traces et erreurs qui vont être décrits en sont une illustration.

4.1 La notion de CHRONOLOGIE

Notre modèle dispose d'une multitude d'évaluateurs potentiels différenciés par un numéro d'ordre de création, un numéro de chronologie. L'accès à cette chronologie est réalisée au moyen de la variable fonction CHRONOLOGY.

Ces évaluateurs peuvent être appelés :

- par l'utilisateur au moyen de la fonction interprète EVAL dont le deuxième argument est le numéro de CHRONOLOGIE que l'on veut voir affecté à cette évaluation :

```
(EVAL  expression  chronologie)
```

- soit automatiquement par l'interprète, il s'agit alors d'interruptions. Ces interruptions sont déclenchées par des événements externes (horloges ou périphériques) ou par l'interprète lui-même dans les cas suivants :
  - appel de la boucle principale du système **(TOPLEVEL)**
  - erreur à l'interprétation **(ERROR)**
  - trace de l'évaluateur **(STEPEVAL)**
  - ligne pleine en sortie **(EOL)**
  - apparition d'une fin de fichier d'entrée **(EOF)**

Le traitement d'une interruption se fait par invocation d'une fonction propre à chaque type d'interruption, dans un nouvel évaluateur dont la chronologie est la chronologie précédente incrémentée d'une unité, i.e. :

```
(EVAL '(fonction associée à l'IT ....) (ADD1 (CHRONOLOGY)))
```

La valeur retournée par la fonction associée à l'interruption devient la valeur de l'interruption.

Il existe de plus une fonction de sortie extra-chronologie, la fonction **EXITCHRONOLOGY** qui réalise la sortie de la chronologie courante et retourne une valeur au créateur de cette chronologie. Cette fonction possède la même syntaxe que la fonction de sortie locale à une fonction, la fonction **EXIT**.

4.2 Les erreurs

Notre modèle ne provoque pas l'abandon du travail en cas de détection d'erreurs. Lorsqu'il se produit des états d'indétermination dans un évaluateur, celui-ci va dynamiquement créer un nouvel évaluateur. Sa tâche est de tenter de lever l'indétermination ou tout au moins de retourner une valeur à l'évaluateur interrompu.

Lorsque ces états d'indétermination sont détectés par un interprète (par exemple lors d'une consultation d'une variable non définie), une fonction spéciale, la fonction **ERROR**, est automatiquement invoquée avec comme arguments les objets indeterminés, dans un évaluateur différent (dont la chronologie est égale à la chronologie antérieure incrémentée d'une unité). La valeur ramenée par cette invocation est utilisée pour lever l'indétermination de l'interprète interrompu.

Cette fonction **ERROR** (comme toutes les fonctions d'interruptions) peut être redéfinie en VLISP afin d'utiliser des sytèmes complexes de corrections automatiques tel que le système PHENARETE de [WERTZ 78], des outils sophistiqués d'aides-à-la mise au point tel que le méta-évaluateur CAN de [GOOSSENS 77, 79], des analyseurs [WERTZ 77] des traces ou des steppers [GREUSSAY 79a]

Voici un exemple de la redéfinition de la fonction **ERROR** qui permet de construire une fonction testant si une forme peut être évaluée sans erreur par EVAL. Dans le cas ou EVAL produirait une erreur cette fonction retourne la valeur NIL, et dans le cas contraire elle retourne la valeur de l'évaluation. Cette dernière valeur est incluse en premier élément d'une liste pour pouvoir distinguer la valeur NIL correspondant à une évaluation de cette même valeur indiquant une erreur à l'évaluation.

```
(DE EVAL-teste-si-erreur (forme)
   (WHERE (ERROR '(() (erreur)))
     (ESCAPE erreur
        [(EVAL forme)])))
```

4.3 Les traces

Une des facilités majeures de notre modèle est d'avoir un mécanisme interne d'observation sélective de l'évaluateur lui-même en fonctionnement. Ce mécanisme permet d'appeler la fonction **STEPEVAL** (qui peut être bien entendu redéfinie en VLISP) avec comme argument la forme qui doit être évaluée, à chaque appel interne de l'évaluateur.

Ce mode trace est activé en donnant une valeur non-NIL au 3ème argument de la fonction interprète EVAL :

```
(EVAL  expression  chronologie  trace)
```

Voici comment réaliser une trace de tous les appels de EVAL.

```
(DE TRACEVAL (exp)
   (WHERE
      (STEPEVAL (forme)
         (PRINT '→ forme)
         (SETQ IT (EVAL forme (SUB1 (CHRONOLOGY)) T))
         (PRINT '← IT))
      (EVAL '(STEPEVAL exp) (ADD1 (CHRONOLOGY)))))

; Fonction de test : dernier élément de l ;

(DE FOO (l)
   (IF (NULL (CDR l)) (CAR l) (SELF (CDR l))))

; Appel de la trace ;

(TRACEVAL '(FOO '(A B)))

→   (FOO '(A B))
→   '(A B)
←   (A B)
→   (IF (NULL (CDR l)) (CAR l) (SELF (CDR l)))
→   (NULL (CDR l))
→   (CDR l)
→   l
```

```
<--   (A B)
<--   (B)
<--   NIL
-->   (SELF (CDR l))
-->   (CDR l)
-->   l
<--   (A B)
<--   (B)
-->   (IF (NULL (CDR l)) (CAR l) (SELF (CDR l)))
-->   (NULL (CDR l))
-->   (CDR l)
-->   l
<--   (B)
<--   NIL
<--   T
-->   (CAR l)
-->   l
<--   (B)
<--   B
<--   B
<--   B
<--   B
<--   B

☞     B
```

Une des difficultés classiques de la réalisation du mécanisme de trace est posée par l'interférence entre la fonction traçante et la fonction tracée en particulier en cas d'utilisation des constructions SELF et EXIT : en effet dans la 11ème ligne de trace l'appel (SELF (CDR l)) doit utiliser la fonction tracée (i.e. la fonction FOO) et non pas la dernière fonction appelée (i.e. la fonction STEPEVAL elle-même). De même l'évaluation d'une forme EXIT peut se rapporter soit au programme traçant soit au programme tracé. Pour lever toute ambiguité à l'évaluation des formes SELF et EXIT, notre modèle utilise la dernière fonction appelée dans la chronologie courante. La fonction traçante et la fonction tracée sont évaluées dans des chronologies différentes, ce mécanisme permettant également de réaliser des traces méta-circulaires *{Note 1}*.

Bien entendu des traces plus élaborées (comprenant des renfoncements correspondants aux niveaux d'imbrications des appels ou des numérotations de lignes par niveau et des possibilitées d'exécution incrémentales) peuvent être construites.

Le fait de pouvoir récupérer TOUS les appels internes de l'évaluateur permet, outre les traces, une modification complète de celui-ci.

{Note 1} KNUTH [KNUTH 69] propose page 211 l'exercice suivant :
"6. [40] Design a trace routine which is capable of tracing itself, in the sense of exercice 4; i.e., it should print out the steps of its own program at slower speed, and that program will be tracing itself at still slower speed, ad infinitum until memory capacity is exceeded."
Notre mécanisme de trace permet de ne tracer qu'un seul niveau de la fonction de trace.

Voici en exemple la fonction **EVAL-NIL-si-UNDEF** qui se comporte comme la fonction interprète standard EVAL mais qui ne provoque pas d'erreur à l'évaluation d'une variable indéfinie, se contentant uniquement de donner la valeur NIL à la variable et d'imprimer un message d'avertissement :

```
(DE EVAL-NIL-si-UNDEF (expression)
   (WHERE
      (STEPEVAL '((forme)
         (IF (OR (LISTP forme) (NUMBP forme) (BOUNDP forme))
             (EVAL forme () T)
             (PRINT "Je donne la valeur NIL à :" forme)
             (SET forme NIL))))
      (STEPEVAL expression)))
```

Voici quelques utilisations de cette fonction en supposant que les variables VARF001, VARF002 et VARF003 sont indéfinies

```
?   (EVAL-NIL-si-UNDEF 'VARF001)
        Je donne la valeur NIL à : VARF001
☞   NIL

?   (EVAL-NIL-si-UNDEF '['A VARF002 'B VARF003])
        Je donne la valeur NIL à : VARF002
        Je donne la valeur NIL à : VARF003
☞   (A NIL B NIL)
```

On notera dans cette fonction la redéfinition dynamique de la fonction **STEPEVAL**, qui permet de n'effectuer le test de variable indéfinie que dans la portée dynamique de la fonction **EVAL-NIL-si-UNDEF**.

5.0 LES SYSTEMES VLISP REALISES.

Une des caractéristiques les plus importantes de notre modèle est d'être très rapidement et très facilement implémenté sur les matériels actuels. Trois réalisations récentes montrent la diversité des machines pouvant recevoir ce système :

1) le système VLISP-10 [CHAILLOUX 78c], est réalisé sur l'ordinateur PDP KI 10 de Digital Equipment Corporation [DEC 78a], sous moniteur TOPS10 6.03 [DEC 78b], SAIL [HARVEY 74, FROST 75] et IRCAM [FROST 77]. C'est le plus rapide et le plus puissant système VLISP existant. Il est disponible dans les centres suivants : l'I.R.C.A.M. et le C.I.T.I.2 en France, et EDIMBOURG, TURIN et STANFORD SAIL à l'étranger.

2) le système VLISP-11 [GREUSSAY 79b], est réalisé sur l'ordinateur PDP 11 également de Digital Equipment Corporation [DEC 75] sous système RT11-V03 [DEC 78c]. Ce système fonctionne actuellement sur les processeurs PDP11 et sur les micro-processeurs LSI11 [DEC 78d] toujours sous système RT11-V03.

3) le système VLISP-8 [CHAILLOUX 79a], est conçu pour le micro-processeur Intel-8080 [INTEL 77a] et Zilog-80 [ZILOG 78]. Il est disponible sous le système de développement ISIS-II [INTEL 77b] et TRS80 Level II [TRS 78].

Chacun de ces systèmes représente une catégorie matérielle spécifique : gros système pour le PDP-10, mini-ordinateur pour le PDP-11 et micro-ordinateur pour le 8080. Les différences entre ces matériels sont considérables. Nous évoquerons les points suivants :

- la taille des mémoires. Le PDP10 permet d'accéder à 256k de 36 bits, le PDP11 à 28k de 16 bits *{Note 1}* et le 8080 à 64k de 8 bits.

- la vitesse d'exécution des instructions. Le transfert dans un registre du contenu d'un mot mémoire demande (indépendemment du nombre de bits transférés) 1.06 micro-sec dans un PDP10-KI, 3.2 micro-sec dans un PDP11/40 et 4.5 micro-sec dans un 8080-2Mhz.

- la puissance des instructions. Les jeux d'instructions de ces machines n'ont pas la même puissance ce qui amène à utiliser un nombre d'intructions plus ou moins grand. Par exemple pour la réalisation de l'allocation et de la construction d'un doublet de liste (le module CONS), il suffit de 2 instructions du PDP10 (qui occupent chacune un mot mémoire de 36 bits) alors qu'il faut un sous-programme de 9 instructions pour le PDP11 (chaque instruction nécessite de 1 à 3 mots de 16 bits) occupant un total de 12 mots pour l'appel et l'exécution du sous-programme. Quant aux performances du 8080 elles sont encore plus mauvaises puisqu'il faut un sous-programme de 21 instructions, qui nécessitent chacune de 1 à 3 mots de 8 bits) occupant un total de 30 mots mémoire.

{Note 1} ce nombre inhabituel de k mémoire (28k) provient du fait que les entrée/sorties sont réalisées en <u>memory mapped I/O</u> sur 4k mots, qui ajoutés aux 28k mémoire conduit bien à un espace adresse de 32k mots.

Malgrés leurs différences, il a été possible d'implanter sur toutes ces machines un système VLISP, tel celui que nous décrire dans cette étude. Ces différents systèmes ne se différencient que par des vitesses d'exécution et des espaces mémoire différents.

Enfin, le champ d'application du système VLISP est extrêmement vaste. Ses principales applications récentes recouvrent les domaines suivants :

- l'amélioration et la correction de programmes avec le système PHENARETE [WERTZ 79].
- la méta-interprétation de programmes récursifs avec le système CAN [GOOSSENS 79].
- les algorithmes d'unification [HULLOT 79].
- la synthèse de programmes à partir d'exemples avec le système SISP [JOUANNAUD 77].
- la compréhension de programmes avec le système RAINBOW [GREUSSAY 79a].
- la synthèse d'images colorées telle qu'elle est pratiquée par le groupe Art et Informatique de l'Université de Vincennes [AUDOIRE 76], [HUITRIC 76]
- l'aide à l'éducation des enfants retardés avec le système LOGO/LISP [WERTZ 79].
- les outils de conception de machines [CHAILLOUX 78b, CHAILLOUX 79b].

BIBLIOGRAPHIE

[ALLEN 78]
ALLEN J. : *The Anatomy of LISP,* Mc Graw Hill Inc., 1978.

[AUDOIRE 76]
AUDOIRE L. : *COLORIX : un périphérique de visualisation couleur,* Mémoire de maîtrise, UER Informatique, Université de Paris 8 - Vincennes, Juin 1976.

[BERKELEY 74]
BERKELEY E. C., BOBROW D. G. (editors) : *The Programming Langage LISP : Its Operation and Application,* The M.I.T. Press, Cambridge, Massachusets, 4th printing, March 1974.

[BOBROW 73]
BOBROW R. J., BURTON R. R., JACOBS J. M., LEWIS D. : *UCI LISP Manual,* Dept. of Information and Computer Science, University of California, Irvine, Technical Report #21 updated 10/73.

[BOLCE 68]
BOLCE J. F. : *LISP/360 : a Description of the University of WATERLOO LISP 1.5 for the IBM System 360,* 2nd ed., University of WATERLOO, Computer Centre, March 1968.

[CHAILLOUX 78a]
CHAILLOUX J. : *VLISP 8 un système LISP pour micro-processeur à mots de 8 bits,* RT 21-78, Département d'Informatique, Université de Paris 8 - Vincennes, Juillet 1978.

[CHAILLOUX 78b]
CHAILLOUX J. : *a VLISP interpreter on the VCMC1 machine,* LISP Bulletin #2, July 1978.

[CHAILLOUX 78c]
CHAILLOUX J. : VLISP *10 . 3 , Manuel de Référence,* RT 16-78, Université de Paris 8 - Vincennes, Août 1978.

[CHAILLOUX 79a]
CHAILLOUX J. : VLISP *8 . 2 , Manuel de Référence,* RT 11-79, Université de Paris 8 - Vincennes, Avril 1979.

[CHAILLOUX 79b]
CHAILLOUX J. : *Etude d'un compilateur* VLISP *optimisant,* Bulletin du Groupe Programmation et Langages, A.F.C.E.T., Division Théorique et Technique de l'Informatique, No. 9 pp. 26-36, Octobre 1979.

[CHAILLOUX 80]
CHAILLOUX J. : *Le modèle VLISP : description, implémentation et évaluation,* Thèse de 3ème cycle, La boratoire Informatique Théorique et Programmation, Université de Paris 6, Avril 1980.

[COILLAND 79]
COILLAND P., COLAITIS M. J. : *SIRLISP : Interprète LISP sur IRIS80,* E.N.S.T., Paris, Février 1979.

[DEC 75]
PDP 11 : Processor Handbook, Digital Equipment Corporation, Maynard Massachusetts, 1975.

[DEC 78a]
DECSYSTEM10 : *System Reference Manual,* Digital Equipment Corporation, Maynard Massachusetts, AD-09 16C-T1, March 1978.

[DEC 78b]
DECSYSTEM10 : *TOPS10,* Digital Equipment Corporation, Maynard Massachusetts, AD-09 16C-T1, March 1978.

[DEC 78c]
RT11 V03 : *Documentation directory,* Digital Equipment Corporation, Order No. DEC 11-ORDDB-A-D, Maynard Massachusetts, 1978.

[DEC 78d]
LSI 11 : *Microprocessor Handbook,* Digital Equipment Corporation, Maynard Massachusetts, 1978.

[DEC 78e]
VAX 11-780, Hardware/Software Handbook, Digital Equipment Corporation, Maynard Massachusetts, 1978.

[DURIEUX 78]
DURIEUX J. L. : *TLISP, le système LISP de TOULOUSE,* in Implémentation et Interprétation de LISP, Ecole IRIA, Toulouse, 1-3 Mars 1978.

[DURIEUX 79]
DURIEUX J. L., VIGNOLLE J. : *Fausses Récursivités dans un Interprète LISP : une approche formelle de leur élimination,* Université Paul Sabatier, Toulouse.

[FROST 75]
FROST M. : *UUO Manual (Second Edition),* SAILON 55.4, Stanford Artificial Intelligence Laboratory, July 1975.

[FROST 77]
FROST M. & HARVEY B. : *The Stanford/IRCAM Monitor,* Institut de Recherche et Coordination Acoustique/Musique, No. 2, October 1977.

[GARDNER 67]
GARDNER M. : *Mathematical Games,* Scientific American, pp. 124-125, March/April 1967.

[GOOSSENS 77]
GOOSSENS D. : *CAN,* Département d'Informatique, Université de Paris 8 - Vincennes, 1977.

[GOOSSENS 79]
GOOSSENS D. : *Meta-interpretation of Recursive List-processing Programs,* Proc. 6th I.J.C.A.I. pp. S7-S12, Tokyo, August 1979.

[GREENBLATT 74]
GREENBLATT R. : *The LISP Machine,* M.I.T. Artificial Intelligence Laboratory, Working Paper 79, November 1974.

[GREUSSAY 72]
GREUSSAY P. : *Manuel LISP 510 : description et utilisation,* Institut d'Intelligence Artificielle, Université de Paris 8 - Vincennes, NTP 2, Octobre 1972.

[GREUSSAY 75]
GREUSSAY P. : *LISP T1600 : Manuel de Référence,* Département d'Informatique, Université de Paris 8 - Vincennes, Février 1975.

[GREUSSAY 76a]
GREUSSAY P. : *VLISP : Structure et extension d'un système LISP pour mini-ordinateur,* RT 16-76, UER Informatique et Linguistique, Université de Paris 8 - Vincennes, Janvier 1976.

[GREUSSAY 76b]
GREUSSAY P. : *Descriptions compactes d'interprètes implémentables : une application au langage CONNIVER,* 2ème Colloque International sur la Programmation, ROBINET B. (éd.), Dunod, Paris, Avril 1976.

[GREUSSAY 76c]
GREUSSAY P. : *Iterative interpretations of tail-recursive LISP procedures,* Département d'Informatique, TR 20-76, Université de Paris 8 - Vincennes, Septembre 1976.

[GREUSSAY 77]
GREUSSAY P. : *Contribution à la définition interprétative et à l'implémentation des LAMBDA-langages,* Thèse, Université de Paris VII, Novembre 1977.

[GREUSSAY 78a]
GREUSSAY P. : *Communication personnelle,* **Octobre 1978.**

[GREUSSAY 78b]
GREUSSAY P. : *Le Système* VLISP *16,* Ecole Polytechnique, Décembre 1978.

[GREUSSAY 79a]
GREUSSAY P. : *Aides à la Programmation en LISP : outils d'observation et de compréhension,* Bulletin du Groupe Programmation et Langages, A.F.C.E.T., Division Théorique et Technique de l'Informatique, No. 9 pp. 13-25, Octobre 1979.

[GREUSSAY 79b]
GREUSSAY P. : VLISP*-11 Manuel de Référence, (à paraître),* Université de Paris 8 - Vincennes, 1979.

[HAFNER 74]
HAFNER C., WILCOX B. : *LISP/MTS programmer's guide,* Mental Health Research Institute, Michigan University, Ann ARBOR, January 1974.

[HAILPERN 79]
HAILPERN B. T., HITSON B. L. : *S-1 Architecture Manual,* Computer System Laboratory, Stanford Electronics Laboratories, Stanford University, Technical Report No. 161, STAN-CS-79-715, January 1979.

[HARVEY 74]
HARVEY B. : *Monitor Command Manual (Second Edition),* SAILON 54.4, Stanford Artificial Intelligence Laboratory, September 1974.

[HEARN 69]
HEARN A. C. : *Standard LISP,* in Bobrow D. G. (ed.) : LISP Bulletin, ACM SIGPLAN Notices, Vol. 4 no. 9, September 1969.

[HEARN 73]
HEARN A. C. : *REDUCE 2 User's Manual,* Utah Computational Physics group UCP-19, Second Ed., March 1973.

[HEARN 74]
HEARN A. C. : *REDUCE 2 Symbolic Mode Primer,* Utah Computational Physics group Operating Note 5.1, October 1974.

[HOLLOWAY 80]
HOLLOWAY J., STEELE G., SUSSMAN G., BELL A. : *The SCHEME 79 Chip,* M.I.T. Artificial Intelligence Laboratory, AI Memo No. 255, Draft, January 1980.

[HORNING 79]
HORNING J. J. : *Additionnal Viewpoints on the History of Programming Languages Conference,* Annals of the History of Computing, Vol. 1 pp. 69-71, July 1979.

[HUITRIC 76]
HUTRIC H. : *Couleur et calcul : calcul de la couleur,* Thèse de 3ème cycle, Université de Paris 8, 1976.

[HULLOT 79]
HULLOT J. M. : *Associative Commutative Pattern-matching,* Proc. 6th I.J.C.A.I. pp. 406-412, Tokyo, August 1979.

[INTEL 77a]
8080 : *Assembly Language Programming Manual,* ON : 98-00301B, Intel Corporation, 1977.

[INTEL 77b]
MDS : *ISIS II System user's guide,* ON : 98-306B, Intel Corporation, 1977.

[JOUANNAUD 77]
JOUANNAUD J. P. : *Sur l'inférence et la synthèse automatiques de fonctions LISP à partir d'exemples,* Thèse, Université Pierre et Marie Curie (Paris 6), Novembre 77.

[KNIGHT 74]
KNIGHT T. : *The CONS Machine,* M.I.T. Artificial Intelligence Laboratory, Working Paper 80, November 1974.

[KNUTH 69]
KNUTH D. E. : *The Art of Computer Programming, Vol. 1 : Fundamental Algorithms,* Addison-Wesley, Reading, Massachusetts, 2nd printing, 1969.

[LAUBSCH 76]
LAUBSCH J. H., KRAUSE D., HESS K., SCHATZ W. : *MACLISP Manual,* CUU-Memo-3, Universitat Stuttgart, Stuttgart, Mai 1976

[LECOUFFE 77]
LECOUFFE P. : *étude et définition d'une machine langage LISP,* Thèse de spécialité, Université des sciences et techniques de LILLE, Décembre 1977.

[LISPMACHINE 77]
LISP MACHINE GROUP, BAWDEN A., GREENBLATT R., HOLLOWAY J. KNIGHT T., MOON D., WEINREB D. : *LISP Machine Progress Report,* M.I.T. Artificial Intelligence Laborartory, Memo No. 444, August 1977.

[LUX 78]
LUX A. : *LISP - IRIS 80 : Manuel d'Utilisation,* Laboratoire IMAG, Février 1978.

[MACSYMA 75]
MACSYMA Reference Manual, Project MAC Mathlab Group, M.I.T., November 1975.

[MARTI 79]
MARTI J., HEARN A. C., GRISS M. L., GRISS C. : *STANDARD LISP REPORT,* ACM SIGPLAN Notices, Vol. 14 No. 10, October 1979.

[McCARTHY 60a]
McCARTHY J. : *LISP 1 Programmer's manual,* M.I.T. Computation Center & Research Laboratory of Electronics, Cambridge Mass., March 1, 1960.

[McCARTHY 60b]
McCARTHY J. : *Recursive Functions of Symbolic Expressions and their Computation by Machine Part I,* C.A.C.M., vol. 3, March 1960.

[McCARTHY 62]
McCARTHY J., ABRAHAMS P. W., EDWARDS D. J., HART T. P., LEVIN M. I. : *LISP 1.5 Programmer's manual,* the M.I.T. Press, Cambridge, Mass., 1962.

[McCARTHY 78]
McCARTHY J. : *History of LISP,* History of Programming Languages Conference, ACM SIGPLAN Notices, Vol. 13 No. 8, 1978.

[MEAD 80]
MEAD C., CONWAY L. : *Introduction to VLSI systems,* Addison-Wesley publishing company, 1980.

[MOON 74]
MOON D. A. : *MACLISP Reference Manual,* M.I.T. Project MAC, Cambridge Mass., April 1974.

[MOORE 76]
MOORE J. S. : *the INTERLISP Virtual Machine Specification,* XEROX Palo Alto Research Center, Palo Alto Cal., September 1976.

[NIVAT 79]
NIVAT M. : *La Recherche en Programmation et ses Moyens de Calcul,* La Gazette du L.I.T.P. Bulletin No. 9, Janvier 1979.

[PERROT 79]
PERROT J. F. : *LISP et lambda-calcul,* Lambda-Calcul et Semantique formelle des langages de programmation B. Robinet, Ed., pp. 277-301, LITP-ENSTA, Paris, 1979.

[QUAM 72]
QUAM L. H., DIFFIE W. : *Stanford LISP 1.6 Manual,* SAILON 28.6, Computer Science Dpt, Stanford University, 1972.

[ROBINET 78]
ROBINET B. : *petit précis de λ calcul,* in Implémentation et Interprétation de LISP, Ecole IRIA, Toulouse, pp. 15-24, 1-3 Mars 1978.

[SCHOETTL 75]
SCHOETTL J. E., WERTZ H. : *Des dragons à foison,* Artinfo/Musinfo #21, Groupe Art et Informatique, Université de Paris 8 - Vincennes, 1975.

[SHIMADA 76]
SHIMADA T. ,YAMAGUSHI Y., SAKAMURA K. : *A LISP Machine and Its Evaluation,* Systems Computers Controls, Vol. 7, No. 3, 1976 (translated from Denshi Tsushin Gakkai Rombunshi, Vol. 59-d, No. 6 June 1976, pp. 406-413).

[STEELE 79]
STEELE G. L. Jr., SUSSMAN G. J. : *Design of LISP-Based Processors or, SCHEME: A Dielectric LISP or, Finite Memories Considered Harmful or, LAMBDA: The Ultimate Opcode,* AI Memo No. 514, M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass., March 1979.

[STOYAN 78a]
STOYAN H. : *LISP,* Thèse, Technishe Universitat Dresden, Dresden, RDA.

[STOYAN 78b]
STOYAN H. : *LISP-Programmier Handbuch,* Akademie Verlag, Berlin, 1978.

[TAFT 79]
TAFT S. T. : *The Design of an M6800 LISP Interpreter,* BYTE, Vol. 4 No. 8 pp. 132-152, August 1979.

[TAKI 79]
TAKI K., KANEDA Y., MAEKEWA S. : *The Experimental LISP Machine,* Proc. 6th I.J.C.A.I., TOKYO, 1979.

[TEITELMAN 73]
TEITELMAN W. : *CLISP - Conversationnal LISP,* Proc. 3rd I.J.C.A.I., pp. 686-690, Stanford, California, August 1973.

[TEITELMAN 75]
TEITELMAN W. : *INTERLISP Reference Manual,* 2nd revision, Xerox Palo Alto Research Center, December 1975, 3rd revision, October 1978.

[TEITELMAN 77]
TEITELMAN W. : *A Display Oriented Programmer's Assistant,* Proc. 5th I.J.C.A.I., pp. 905-915,Cambridge, Mass., August 1977.

[TRS 78]
TRS80 : *LEVEL II,* Radio Shack, Ft. Worth, Texas 76102

[WEINREB 79]
WEINREB D. & MOON D. A. : *LISP MACHINE MANUAL,* M.I.T. Cambridge, Mass., January 1979.

[WEISSMAN 67]
WEISSMAN C. : *LISP 1.5 Primer,* Dickenson Publishing Company Inc., Belmont, California, 1967.

[WERTZ 74]
WERTZ H. : LISP CAB500, *Rapport de Recherche goupe II équipe 9,* Université de Paris 8 - Vincennes, 1974-1975.

[WERTZ 77]
WERTZ H. : *VLISP - AID,* Université de Paris 8 - Vincennes, RT 10-77, Juillet 1977.

[WERTZ 78]
WERTZ H. : *Un système de compréhension, d'amélioration et de correction de programmes incorrects,* Thèse de 3ème cycle, Université Paris VI, Juillet 1978.

[WERTZ 79]
WERTZ H. : *Computer Aided Education for Mentaly Retarded Children,* Proc. International Symposium on Computer and Education, Dusseldorf, RFA, Mars 1979.

[WHITE 76]
WHILE J. L. : *Mail from JONL at MIT-ML to LISP-discussion at MIT-AI,* February 1976.

[WHITE 78]
WHITE J. L. : *Programm is Data,* History of Programming Languages Conference, ACM SIGPLAN Notices, Vol. 13 No. 8 pp. 217-223, 1978.

[WINSTON 77]
WINSTON P. H. : *Artificial Intelligence,* Addison-Wesley Company Inc., 1977.

[ZILOG 79]
Z80 : CPU Technical Manual, Zilog Inc., 1979.

INDEX BIBLIOGRAPHIQUE